Working with JavaScript Objects and Events

Enhancing your Forms with JavaScript

Validating User Input in the Neonatal Feeding Study

St. Mary's hospital, located in the city of Northland Pines, is a large complex serving the needs of the community and neighboring towns. St. Mary's is also a research hospital, which means that it is the home of several research institutions, including the Midwest Clinical Cancer Center and General Clinical Research Center.

One of the tasks involved in clinical research is enrolling patients in various studies. Each patient in a study must go through a registration process, including the completion of a registration form used to determine whether the patient is eligible to participate.

In past years, these forms were paper records, filled out by hand by the attending nurse or physician. Recently, however, the hospital has developed an intranet. Some researchers want to place registration forms on the hospital intranet, allowing them to be filled out online and automatically sent to a database for storage.

You've been asked to help develop one of the first online registration forms, for study run by Dr. Karen Paulson on the effects of different feeding methods on newborn infants. The form will have to be interactive in that it will have to calculate key items as well as check the user's entries for mistakes. To create this Web page form, you'll have to use JavaScript to control how users access and enter data into the form.

Understanding Form Validation

Session 1

In this session you will learn

- About the principles of form validation
- How validation applies to the objects found on a Web page form
- About the object-based nature of the JavaScript language and explore the principles of objects, properties and methods.

Open the file Studytxt.htm in your text editor and save it as Study.htm in the same folder.

Open the file Study.htm in your browser.

Note that the name of the form is "REG"

This form collects basic info on newborns

As the form is currently designed, almost any value could be entered into any of the input fields. This concerns Dr. Paulson and to address her concerns, she explains what she would like the online form to do automatically for the user.

- The Date field should be completed automatically for the user, to avoid the possibility of an incorrect date being entered. The user should still be able to change the date, if necessary.

- If "Other" is selected from the Physician selection list, the user should be prompted to enter the physician's name in the "If other" input box. Otherwise, the insertion point should jump directly to the APGAR scores when the user presses the Tab key.

- The 1-minute total APGAR score should be calculated automatically based on the value of its 5 component parts.

- The value of each component of the APGAR score (activity, pulse, grimace, appearance and respiration) can be only 0, 1, or 2. The form should reject all other values and alert the user that an improper value was entered.

- No registration form should be submitted unless a parental consent form has been filled out, as indicated by a check in the Parental Consent check box on the registration form.

To meet Dr. Paulson's requirements, your Web page form must be able to react in different ways, depending on the user's input. The form must be able to skip certain fields if the user selects a certain item, but not if the user selects a different item. The form must also check the APGAR value that the user enters and either calculate a total score or, if necessary, inform the user that a mistake has been made.

Dr. Paulson's criteria are examples of form validation, a process by which the server or the browser checks form entries, and where possible errors are eliminated. On the Web, form validation can occur on either the client side or the server side.

In client-side validation, the form is checked as the user enters the info and immediate feedback is provided if the user makes a mistake.

One powerful use of JavaScript is to provide this kind of client-side validation. With a script built into the Web page form, you can give immediate feedback to users as they enter data, which also reduces the amount of network traffic between users and the Web server. Before you can do this for Dr. Paulson's form, however, you must first learn how JavaScript can be used to manipulate elements on your Web page, not just when the page is initially loaded, but also in response to events initiated by the user. The first step in accomplishing this is to understand the object-based nature of the JavaScript language and how it can be used to control the behavior of the Web page, the form on the page and even the Web browser itself.

Working with an Object-based Language

JavaScript is an object-based language, which means that the language is based on manipulating objects by either modifying their properties or applying methods to them. Objects are items that exist in a defined space on a Web page. Each object has properties that describe its appearance, purpose or behavior. Furthermore, an object can have methods, which are actions that can be performed with it or to it.

Consider the example of an oven in your kitchen. The oven is an object. It has certain properties, such as its model name, age, size and color. There are certain methods you can perform with the oven object, such as turning on the grill or the self-cleaner. Some of these methods change the properties of the oven, such as the oven's current temperature. You modify the oven's temperature property through the method of turning the stove on or off.

Similarly, your Web browser has its own set of objects, properties and methods. The Web browser itself is an object, and the page you're viewing is an object. If the page contains frames, each frame is an object and if the page contains forms, each field on the form (as well as the form itself) is an object. These objects have properties. The browser object has the type property (IE, or Opera), the version property (3.0, 4.0 or 5.0). There are some methods you can apply to your browser: you can open it, close it, reload the contents of the browser window or move back and forth in your history list.

Understanding JavaScript Objects and Object Names

An object is identified by its object name, a name that JavaScript reserves for referring to a particular object.

When you want to use JavaScript to manipulate the current window, for e.g. you have to use the object name "window". Operations that affect the current Web page use the"document" object name.

An object can also use a name that you've assigned to it. You've seen many HTML tags that include the name property, such as the <form>, <frame> and <input> tags. You can refer to objects created from those tags with the values specified in the name property. For example, in Dr. Paulson's registration form, the following tag starts the form:

        <form name=REG>

You refer to this form using the object name, REG, in your JavaScript program.

JavaScript Object Hierarchy

The topmost object in the hierarchy is the window object. The window object contains the other objects in the list, such as the current frame, history list or document. The document contains its own set of objects, including links, anchors, forms and within each form, forms elements such as input boxes.

In some situations, you will need to specify this hierarchy when referring to an object. You can do that by separating each object by a period and including the objects in the object name, starting at the top of the hierarchy and moving down. For example, in Dr. Paulson's form, the MEDRECNO input box lies w/in the REG form, which lies w/in the application window. The complete object reference for the input box, including the other objects in the hierarchy is:

        window.document.REG.medrecno

In most cases, you can omit the window object name from the hierarchy and JavaScript will assume that it is there. In other words, JavaScript treats the above object reference in the same way as the following reference:

        document.REG.medrecno

When working with objects on Web page forms, such as the form you'll be developing for Dr. Paulson, you should include the entire hierarchy of object names (except for the window object).

Working with Object Properties

Each object in JavaScript has properties associated with it. The number of properties varies depending on the particular object. Some objects have only a few properties whereas others have dozens. As with object names there are certain keywords that identify these properties.

There are several ways of working with properties. You can change the value of a property, store the property's value in a variable or test whether the property equals a specified value in an If…Then expression.

Modifying a Property's Value

The syntax for changing the value of a property is:

        object.property = expression

where object is the JavaScript name of the object you want to manipulate, property is a property of that object and expression is a JavaScript expression that assigns a value to that property.

Not all properties can be changed. Some properties are read-only which means that you can read the property value but you can't modify it. One such property is the appVersion property of the navigator object, which identifies the version number of your Web browser. Although it

would be nice to upgrade the version of your browser by using a simple JavaScript command, you're not allowed to change this value.

In this example above, the values of the appCodeName, appName and appVersion properties are used to display the browser code name, browser name and version on the Web page. You might use this info when creating pages that involve HTML extension supported by specific browsers or browser versions. Your JavaScript program could first test to see whether the user is running one of those browsers before inserting the tags into the Web page.

Assigning a Property to a Variable

Although you cannot change the value of read-only property, you can assign that value to a variable in your JavaScript program. The syntax is:

    variable = object.property

where variable is the variable name, object is the name of the object and property is the name of its property.

Working with Object Methods

Another way of controlling your Web page is to use methods. Recall that methods are actions that objects can perform or actions that you can apply to objects.

The syntax for applying a method to an object is:

    object.method (parameters);

where object is the name of the object, method is the method to be applied, and parameters are any values used in applying the method to the object. If you are using multiple parameters, commas should separate each of these. One of the most commonly used methods is the write() method applied to the document object, which sends text to the Web page document.

.

Session 2

In this session you'll learn

- About events in JavaScript
- How to run JavaScript programs in response to specific events
- How to initiate these events from w/in a program
- Validate a Web page form
- How to prompt the user for info
- How to alert the users when mistakes have been made

The following is a checklist of changes that Dr. Paulson wants you to make to her form.

- Automatically enter the current date in the formdate field and move the cursor to the next field in the form
- If "Other" is selected from the Physician selection list box, prompt the user for the name of the physician, otherwise go to the APGAR component fields
- Automatically calculate the total APGAR score
- Check that valid APGAR component scores have been entered.
- Check that parental consent has been obtained before submitting the form.

Your first task is to set up the form so that the current date is entered automatically into the formdate field whenever the browser opens the page. The action of the user opening the form is an example of an event. An event is a specific action that triggers the browser to run a block of JavaScript commands.

Events can take place in rapid succession. Consider the example where a user presses the Tab key to enter text into an input field, changes the field's value and leaves the field by pressing the Tab key. The first event that the browser recognizes in this scenario is the Focus event, as the input field becomes the active filed in the form. After the user changes the value in the field and leaves the field, the Change event is triggered as the browser notes that the value of the field has been changed. Finally, the blur event occurs as the focus leaves the field and goes to a different field on the form.

With so many different events associated with your Web objects, you need some way of telling the browser how to run a set of commands whenever a specific event occurs. This is where an event handler becomes important.

Using Event Handlers

An event handler is code added to an HTML tag that is run whenever a particular event occurs w/in that tag. The syntax for invoking an event handler is:

> <tag event_handler ="JavaScript commands;">

where tag is the name of the html tag where the event occurs, event_handler is the name of an event handler and JavaScript commands are the set of commands or more often a single command that calls a JavaScript function to be run when the event occurs. Different html tags have different event handlers.

Using the onload Event Handler

Now that you've learned about objects, properties, methods and events, you are ready to start modifying Dr. Paulson's form.

The event handler for opening of a Web page is called onload. Because this handler is associated with the document object, you must place it in the <body> tag of the html file. The event handler should run the function StartForm(). StartForm() is a user-defined function that you" create shortly.

Start your text editor and open the Study.htm file

Type onload="StartForm();" in the <body> tag of the HTML

Save your changes

Now you have to create the StartForm() function. This function will have 2 purposes:

First it will enter the current date into the Date field, and then it will move the cursor to the next field in the form

JavaScript functions are usually collected together between a set of <script> tags located in the head section of the file. The StartForm() function relies on another JavaScript function named DateToday(), which has already been created for you in the Study.htm file. The code for the DateToday() function is as follows:

```
function DateToday() {

        var Today=new Date();

        var ThisDay=Today.getDate();

        var ThisMonth=Today.getMonth();+1

        var ThisYear=Today.getFullYear();

return ThisMonth+"/ "+ThisDay+"/ "+ThisYear;
}
```

This function contains JavaScript commands that you should be familiar with. It uses the date object and extracts the current day, month and year and then combines those values in a text string, which it then returns to the user. StartForm() will call the DateToday() function to retrieve a text string containing the current date and then it will place that text in the Date field in the registration form. The code for the StartForm() function is as follows:

```
function StartForm() {

        document.REG.formdate.value=DateToday();

}
```

The program calls the DateToday() function which creates a text string displaying the current date. The value of the formdate field of the REG form in the current document is then made equal to this text string. In other words, the current date is displayed in the formdate field, which is what Dr. Paulson wants.

Add the StartForm() function to the Study.htm file:

Go to the <script> tag in the Study.htm file

Below the closing bracket of the DateToday() function enter the following commands:

```
function  StartForm()  {

        document.REG.formdate.value=DateToday();

    }
```

Save your changes and your Study.htm page should show the current date upon reloading.

The function does one of the 2 jobs it needs to; it displays the date in the Date field. Dr. Paulson also wants the function to move the cursor to the next field in the form. To accomplish this, you have to learn how to make JavaScript not only respond to an event but also initiate one.

Emulating Events

When you use JavaScript to emulate an event, you are causing the Web page to perform an action for the user, such as having the cursor move to the next field. To emulate an event, you apply an event method to an object on your Web page.

You need to add a command to the StartForm() function that places the cursor in the next field in the REG form, which in this case is the firstname field. The command to move the cursor to the firstname field is:

```
document.REG.firstname.focus();
```

Locate the StartForm() function at the beginning of the file and then add the following command to the end of the command block

```
document.REG.firstname.focus()
```

Save you changes and reload the page and the focus should now be at the firstname field.

You have completed your first task on your list for Dr. Paulson.

The next few fields in the form don't require any modifications. However, Dr. Paulson does want you to change the behavior of the Physician selection list.

Working with a Selection Object

Dr. Paulson wants the form to control how the user selects a physician from the selection list. If the user selects one of the seven physicians in the list, the cursor should immediately go to the field for the APGAR Activity score, skipping over the If other field. On the other hand, if the user selects "Other" from the list of physicians, the form should prompt the user to enter that physician's name before continuing.

You need to create another function to handle this task. The function, which you'll name CheckOther(), should run whenever the cursor leave the Physician selection list. This action of leaving a field is managed by the onblur() event handler. You'll now add this event handler to the <select> tag for the Physician selection list.

Add the onblur() event handler to the tag:

Return to your text editor and the Study.htm file

Locate the <select> tag for the Physician selection list and then insert the following code into the tag:

    onblur="CheckOther();"

Now you need to create the CheckOther() function. To do so, you need to learn a little about how JavaScript works with selection lists and their options. JavaScript treats a selection list as an array of option values. In the case of Dr. Paulson's form, the tags that define the Physician selection list are:

```
<select name=physician>
        <option value="Albert">Dr. Warren Albert
        <option value="Alvarez">Dr. Maria Alvarez
        <option value="Brinkman">Dr. Karen Brinkman
        <option value="Kerry">Dr. Michael Kerry
        <option value="Nichols">Dr. Chad Nichols
        <option value="Paulson">Dr. Karen Paulson
        <option value="Webb">Dr. Tai Webb
        <option value="Other">Other
</select>
```

Each option in the selection list has a value property that corresponds to the VALUE property entered into the <option> tag. For the Physician selection list, the following are the JavaScript objects and properties for each option value:

```
document.REG.physician.options[0].value="Albert"
document.REG.physician.options[1].value="Alvarez"
document.REG.physician.options[2].value="Brinkman"
document.REG.physician.options[3].value="Kerry"
document.REG.physician.options[4].value="Nichols"
document.REG.physician.options[5].value="Paulson"
document.REG.physician.options[6].value="Webb"
```

document.REG.physician.options[7].value="Other"

Each option in the selection list belongs to a hierarchy of object names. In this case, the hierarchy starts with the document object, goes to the REG form w/in the document, then goes to the physician field w/in the form and finally goes to each individual option w/in the selection list. Note that the array of selection options starts with an index value of 0 and not 1.

Similarly, the text that actually appears in the selection list is specified by the text property. For options in the Physician selection list, this results in the following objects and properties:

document.REG.physician.options[0].text="Dr. Warren Albert"
document.REG.physician.options[1].text="Dr. Maria Alvarez"
document.REG.physician.options[2].text="Dr. Karen Brinkman
document.REG.physician.options[3].text="Dr. Michael Kerry"
document.REG.physician.options[4].text="Dr. Chad Nichols"
document.REG.physician.options[5].text="Dr. Karen Paulson"
document.REG.physician.options[6].text="Dr. Tai Webb"
document.REG.physician.options[7].text="Dr. Other"

The first task the CheckOther() function should perform is to determine whether the user has chosen the option "Other" from the Physician selection list. The option the user selects is stored in the selectedIndex property of the selection list.

The full reference for the physician selection list is:

document. REG.physician.selectedIndex

For example, if the user selects "Other" from the selection list, the selectedIndex property has a value of 7, because Other is the seventh item in the array of physician options.

If the user selects "Other" from the list of physicians, you want the CheckOther() function to prompt the user for the physician's name before going onto the next field in the form (the ACT field). However, if the user selects one of the physicians in the list, then the form should proceed to the ACT field without prompting.

Add the CheckOther() function to the Study.htm file:

Go to the <script> tag in the Study.htm file

Below the StartForm() function, enter the following commands:

```
function CheckOther() {

        if (document. REG.physician.selectedIndex==7) {

                //Prompt for the name of the physician

        }
```

```
document.REG.act.focus();

}
```

Save your changes and reload the file in your browser and note when one of the physician is selected from the list and TAB is pressed the cursor moves to the act field.

The first line of command block checks whether the Other option (index value 7) is selected. The second line of the command block, which is only a comment right now, will prompt the user to enter the name of the doctor in the Other field. This will be covered shortly. The last line of the command block, the Else statement causes the cursor to move to the Activity field if the index value of 7 was not selected.

Prompting the User for Input

To prompt the user for input, you use the prompt() method. The prompt() method creates a dialog box containing a message you create and an input field into which the user can type a value or text string. The syntax for the prompt() method is:

```
prompt("Message", "Default_text");
```

where Message is the message that you want to appear in the dialog box, and Default_text is the default text that you want to appear in the dialog box's input filed.

Note that different browsers will display their dialog boxes slightly differently; however, all dialog boxes will share the common features of a title bar, default value, OK button and Cancel button.

The prompt() method also returns a result that can be stored in a variable or placed in an object. For example, the following JavaScript command will place whatever text the user enters in the dialog box into the UserName variable:

```
UserName=prompt("Enter your name", "Type name here");
```

You can use the prompt() method in the CheckOther() function to prompt the user for the name of the physician and then to insert the response into the othername field.

Add the prompt() method to the CheckOther() function:

Replace the comment. "//Prompt for the name of the physician" with the following:

document.REG.othername.value=prompt("Enter name of physician", "Name");

Save your changes and reload the file

Select Other and press TAB

Enter the name Dr. Carol White and click Ok. Observe what happens.

Session 3

In this session you will learn how to:

- Create a calculated field
- Perform validation checks on values that the user enters
- Notify the user when mistakes have been made
- Perform a final validation check when the form is submitted.

Creating Calculated Fields

Next task is to calculate the total APGAR score automatically and store this value in the TOTAL field. The APGAR score is a measure of the general health of a newborn baby and has 5 components: activity, pulse, grimace, appearance and respiration. Each component is given a score of 0, 1 or 2 and the formula for the total APGAR score is simply the sum of all 5 components:

Dr. Paulson wants the form to recalculate the total APGAR value every time the user enters a value in one of the APGAR fields.

The first thing you'll do then, is add the onblur() event handler to each of the 5 APGAR fields in the form. Every time the user tabs out of the one of these fields, the Web browser will call a function (which we will create shortly) that calculates the current APGAR total. The total value will then be stored in the total field on the form.

Add the onblur() event handler to the five APGAR fields:

Reopen Study.htm

Go to the <input> tag for the Activity field and insert w/in the tag the following onblur event handler

onblur="APGAR();"

APGAR() is the name of the JavaScript function that you'll create to calculate the total APGAR score and then store the result in the total field.

Add this same line of code to the <input> tags of the other 4 component fields (but not to the total field)

Save your changes

Next, you need to create the APGAR() function. This function should add up the values entered into each of the component fields; however, JavaScript has a quirk you have to account for.

JavaScript treats values entered into input boxes as text strings, so you must first convert them from the text format to the number format. Otherwise the APGAR() function will produce an error.

You can change text to numbers with the eval() function.

One feature of the eval() function is that it takes a number that is represented as a text string and converts it to a number. For e.g. the following command takes the text string "10," converts it to the number 10 and stores that value in the variable total:

        total = eval("10");

Converting the text values to numbers is important; otherwise, JavaScript will simply append one text string to the other. For e.g., consider the following command:

        "10" + "5"

This command produces the text string "105" because you're adding 2 text strings together, rather than the values the text strings represent. However, the following command results in the numeric value 15:

        eval("10") + eval("5")

Using the eval() function, you would create the APGAR() function as follows:

Create the APGAR() function:

Locate the CheckOther() function at the top of the Study.htm file and insert the following lines of code below it.

```
function APGAR() {
        var A = eval(document.REG.act.value);
        var P = eval(document.REG.pulse.value);
        var G = eval(document.REG.grimace.value);
        var AP = eval(document.REG.app.value);
        var R = eval(document.REG.resp.value);
        document.REG.total.value=A+P+G+AP+R;

}
```

Save your changes.

Enter the following APGAR component values (press the Tab key to move between the fields):

        2 in the Activity field
        0 in the Pulse field
        1 in the Grimace field
        2 in the Appearance field
        2 in the Respiration field

Each time you enter a component value and press the Tab key to leave the field, the onblur() event handler is triggered and the APGAR() function is run, updating the value in the total field.

By using the onblur() event handler and the APGAR() function, you've created a field that is automatically calculated for the user. In its current form, the APGAR() function will accept any

value for each of the components; however, the function must also make sure that only valid APGAR component scores are entered. You'll make this change next.

Validating User Input

Dr. Paulson wants the form to allow only the values 0, 1 and 2 to be entered into the APGAR component fields; these are the only values physician assign when performing an examination of a newborn. If the user enters an incorrect value 2 things must happen:

1. The browser should display a dialog box informing the user of the mistake.

2. The cursor should be positioned back in the field in which the user entered the incorrect value, preventing the user from leaving the field until a valid value has been entered.

This presents a problem in the registration form. Five different component fields could be calling the APGAR() function. How do you know which field is the one in which the user entered an incorrect value? To make this work, you have to pass info to the function, indicating which field is using it.

You do this with the "this" keyword.

The "this" Keyword

The this keyword is a word reserved by JavaScript to refer to the currently selected object, whatever that might be. For e.g., if the Pulse field is the current field, the following 2 commands produce the same action (changing the value of the Pulse field to 2).

    document.REG.pulse.value = 2;

    this.value = 2;

You can also use the "this" keyword to pass info about the currently selected field to a function. For e.g., assume that you have 2 input boxes, pulse and resp both of which will use the same function, setVal(), as shown below:

```
<script type="text/javascript">

        function setVal(field) {

                field.value = 0;

        }

</script>
```

<input name = pulse onfocus="setVal(this);">

<input name = resp onfocus="setVal(this);">
When the pulse input box receives the focus, it calls the setVal() function, including the "this" keyword, as a parameter value. The setVal() function is then applied to the currently selected

object. When the pulse field is the currently selected object, the setVal() function changes the pulse value to zero. When the resp field is the currently selected object, its value is similarly set to zero.

In the same way, you can modify the APGAR() function to include info, through the "this" keyword, to indicate which field is calling the function. You can test then whether the value of the field violates the rules that Dr. Paulson has set up for APGAR scores.

Add the "this" keyword to the onblur() event handlers:

Return to the Study.htm file and go to the <input> tag for the Activity field.

In the code onblur="APGAR();" type the word this within the parentheses so that the code now reads, onblur="APGAR(this);"

Repeat the above step for the remaining 4 components of the APGAR scores

Save your changes.

Now that you've added the "this" keyword to the event handler that calls the APGAR() function, you need to make several changes to the function itself.

1. The function must store the value of the active field in a variable.

2. It needs to test whether or not the value of that variable is equal to 0, 1 or 2. If it is the function can calculate the total APGAR score as before, if not, the function should alert the user that a mistake has been made and return the cursor to the appropriate field so that the user can enter the correct value.

The revised APGAR() function should be as follows:

```
function APGAR(field) {

if(field.value==0 || field.value==1 || field.value==2) {

        var A = eval(document.REG.act.value);
        var P = eval(document.pulse.act.value);
        var G = eval(document.REG.grimace.value);
        var AP = eval(document.REG.ap.value);
        var R = eval(document.REG.resp.value);
        document.REG.total.value = A + P + G + AP + R;

} else {

        //alert the user

        field.focus();

}
}
```

The new version of the APGAR() function has a single parameter, "field", which records which field called the function. Then the function tests whether the field value equals 0, 1 or 2 (remember that the || symbols represent the "or" logical operator). If this is the case, the function calculates the total APGAR score as before and the user continues to the next field in the form. However, if the field's value is other than 0,1 or 2, the function should alert the user that a mistake has been made and return the user to the field using the focus() method. The command for alerting the user is inserted as a comment at this point. We will modify this with a dialog box soon.

Revise the APGAR() function:

Locate the APGAR() function at the top of the Study.htm file

Change the function line function APGAR() to function APGAR(field)

Add the following 2 lines below the function statement as shown

if(field.value==0 || field.value==1 || field.value==2) {

Insert the following commands above the last line (the closing braces}) of the function

} else {

//alert the user

field.focus();

Save your changes.

Notifying the User with Alert and Confirm Dialog Boxes

If the user enters an incorrect value for one of the APGAR components, the form should display a dialog box informing the user of the error. To accomplish this, you can use the alert() method. The alert() method operates in the same way as the prompt() method, except that it does not provide an input box in which the user can type a response. Instead, it simply displays a dialog box containing a message. The syntax for the alert() method is:

    alert("Message");

Different browsers display slightly different dialog boxes.

JavaScript provides another method called the confirm() method, which works in the same way as the alert() method, except it has 2 buttons on the dialog box, an OK and Cancel button. If the user clicks the OK button, the value "true" is returned and "false" if cancel is clicked.

You would use the confirm() method in situations that require a simple "yes" (OK) or "no" (Cancel) response from the user.

Add the alert() method to the APGAR() function:

In the APGAR() function, replace the comment "//alert the user" with the following line of code:

       alert("You must enter a 0, 1 or 2");

Save your changes and reload the file in your browser and test whether the alert functions when an incorrect value is entered.

Controlling Form Submission:

Dr. Paulson wants the registration form to perform a validity check to determine whether or not a parental consent form has been filled out. This is indicated on the registration form by the Parental Consent check box. If the box is checked, then it is assumed that the consent form has been filled out. This validity check must be performed when the user tries to submit the form.

When a user completes a form and then clicks the Submit button, a Submit event is initiated. JavaScript provides the onsubmit event handler to allow you to run a program in response to this action. Because the Submit event is associated with the form object, you must place the event handler in the <form> tag as shown below:

```
<script type="text/javascript">
        function goodbye() {
        alert("Thank you for your time");
        }
</script>

<form onSubmit="goodbye();">
```

In this example, the goodbye() function is run automatically when the user clicks the Submit button located elsewhere in the HTML file and a dialog box with the message "Thank you for your time" appears. This is a simple example in which the function does not actually perform any validation; it just displays a message. When you need to validate the form or a particular field in it, the situation is a bit different.

The syntax for validating your form before submitting it is:

       <form onsubmit="return function_name();">

where function_name is the name of the function that is used to validate your form.

The function must return a value of either "true" or "false". If the function value is "true", the form will be submitted to the CGI script. If the value is "false", submission is canceled and the user is returned to the form (presumably to correct the problem). Note the inclusion of the keyword "return" in this command. The return keyword forces the browser to apply the results of the validation function. If you do not include the return keyword, the browser will submit the form whether or not it passes the validation test.

For e.g. if you create a function named Check_Data() to validate your form, the correct version of the onsubmit event handler will be:

<form onsubmit="return Check_Data();">

and not:

    <form onsubmit="Check_Data();">

Add the Check_Data() function to the onsubmit event handler:

Return to the Study.htm file and go to the <FORM> tag (located below the list of functions)

Within the <form> tag, insert the following command

    onsubmit="return Check_Data();"

Save your changes

Now you need to create the Check_Data() function. The purpose of this function is to simply determine whether or not the Parental Consent check box has been checked.

You can tell whether a check box object has been checked by using the "checked" property. It is a Boolean property either true or false. In Dr. Paulson's form the Parental Consent check box has the field name consent. So if,

    document.REG.consent.checked

equals "true" then the check box has been checked. If the property value is "false" the check box has been left unchecked and the form should not be submitted because no parental consent has been given.

The Check_Data() function will test which of these 2 conditions has occurred. If the check box has been selected, the function will alert the user that the form has been completed successfully and will return a value of "true". If the check box has not been selected, the function will alert the user of the problem and return a value of "false".

Create the Check_Data() function:

Below the APGAR() function at the top of the Study.htm file, insert the following lines:

    function Check_Data() {

            if(document.REG.consent.checked==true) {

                    alert("Form completed successfully");

                    return true;

            } else {

                    alert("You still need parental consent");

```
                return false;
        }

    }
```

Save your changes to Study.htm and reload the file in your browser.

Now you can test the Check_Data() function by trying to submit the form w/out first checking the Parental Consent check box.

In a more advanced version of this page, you might add other variation criteria to the Check_Data() function. For e.g., you might check whether or not a name and medical record number have been entered for the patient. You might double-check the values of each APGAR component to verify that they still valid. In that case the Check_Data() function would include several If…Else conditions.

Reloading a Page with the Location Object

There is one more issue to consider with the registration form. A user who wants to reset the form can press the Reload button located next to the register button. The Reload button resets all of the fields in the form to their default values. Is this what Dr. Paulson wants? Not exactly; recall that the first action this form takes is to insert the current date into the formdate field. This action runs whenever the page is loaded. Unfortunately, resetting a form is not the same as reloading the page. The date value would not be entered automatically if you simply reset the form. Instead, your form should actually be reloaded, an action which includes the onload event, which runs the StartForm() function, which in turn inserts the current date in the form.

To reload a page, you use the location object. The location object indicates the location of the page in the browser. To reload the page, use the reload() method.

```
        location.reload();
```

You also can use JavaScript to access a page in a different location. The following command:

```
        location=URL
```

will cause the browser to load a Web page with the address, URL.

Because the command for reloading a page is a single-line command, you can enter it directly into the <input> tag for the Reload button. The command should be activated whenever the button is clicked, so you'll use the onclick event handler in the tag

Insert the command to reload the page:

Return to the Stuy.htm file and go to the <input> tag for the Reload button at the bottom of the form. This tag is located in the section "Form registration and reset buttons"

Insert the following command into the <input> tag

onclick="location.reload();"

Save your changes to the Study.htm file and reopen the file in your Web browser.

Enter some text in the form and click the Reload button. Verify that the page reloads properly: all the fields should be reset to their default values, the current date should be inserted into the formdate field and the cursor should be in the first name input box.